Representing Concepts for Automated Reasoning with Natural Language

Tanel Tammet[1 [0000−0003−4414−3874]], Priit Järv[1 [0000−0001−7725−543X]], Martin Verrev[1 [0000−0003−4890−9283]], and Dirk Draheim[2 [0000−0003−3376−7489]]

[1] Applied Artificial Intelligence Group,
Tallinn University of Technology, Tallinn, Estonia
{tanel.tammet,priit.jarv1,martin.verrev}@taltech.ee
[2] Information Systems Group, Tallinn University of Technology, Tallinn, Estonia
dirk.draheim@taltech.ee

Abstract. The focus of the paper is on the representation of commonsense concepts in natural language for automated reasoning in first order predicate logic (FOL). The context of our work is query answering with hybrid neurosymbolic reasoning, relevant for the domains requiring control, tool use, trustworthiness or explainability. The semantic parser generates enhanced FOL representations of the query and the background knowledge provided. We experiment with using a large language model (LLM) in various roles of a general pipeline consisting of a semantic parser and an automated reasoner. The FOL representation is then solved using an automated reasoning engine enhanced with algorithms for commonsense reasoning. The representation scheme proposed is geared towards efficient proof search, while retaining sufficient generality for encoding a large class of problems.

1 Introduction

Semantic parsing – converting natural language to a symbolic representation – is one of the classical areas of artificial intelligence. Our paper considers different ways to encode concepts stemming from semantic parsing to enhanced first order logic, with a focus on improving efficiency of automated reasoning while keeping the encoding sufficiently simple for human control and understanding.

Although a large body of research has been devoted to semantic parsing, the practical use has been limited: one of the reasons is the complexity of adequately interpreting natural language, while the other two are the limited performance of reasoning systems on the parser output and severe limitations of the existing knowledge bases the reasoning engines can use.

The advent of large language models (LLMs) has seemingly diminished the interest in semantic parsing: indeed, the performance of LLMs on most of the related benchmarks for question answering and commonsense reasoning by far outperforms the systems built around semantic parsers. On the other hand, LLMs are increasingly used as tools for semantic parsing, with arguably better results than achieved by using classical semantic parsing techniques.

While there is little doubt that the performance of LLMs on language-focused tasks is superior to other currently available techniques, LLMs are known to have a number of significant limitations: erroneus inferences (hallucinations), lack of explainability and controllability, complexity of adding new information or modifying the information the LLM has been trained on. All of this hampers their use in applications requiring rigor, control and large amounts of often-changing data: i.e. most of the "ordinary" IT systems. There are currently no grounds to predict whether these problems will be mostly overcome in the future versions of LLMs, or new paradigms are needed.

Reagrdless of the issues mentioned above, it is clear that LLMs are conceptually not well suited for problems requiring search, like planning (see [31]), theorem proving or finding long chains of inferences. Simply put, LLMs do not contain algorithms for search. The paper [6] presents recent experiments and a discussion, stating "we offer novel insights into the fundamental limits of transformers, centered around compositional problems that require strict multi-hop reasoning to derive correct predictions."

A growing body of research, e.g. [30], [14], [19] and [8], is devoted to using LLMs as specialized semantic parsers producing symbolic input for conventional software systems such as planners or programming language interpreters, which are then employed to solve the tasks originally posed in natural language. These systems, in turn, may gain performance from components using machine learning, which are specialized for the task.

Our approach to semantic parsing is to generate a representation which is usable for a wide spectrum of query answering and planning tasks, either directly or via further transformations. We use enhanced first order logic for symbolic representation and verify the generation of representation by using an enhanced first order logic theorem prover for question answering.

As a general rule, specialized representations are more efficient and easier to use for specialized tasks than general representations, like first order logic. However, in situations where a large body of text has to be represented symbolically for various kinds of queries and tasks to be asked later – for example, in the context of a dialog system – it may be unfeasible to generate a suitable specialized representation beforehand.

Another view on the same issue is the question of whether it would be advantageous or necessary for further A.I. progress to use structured world models: essentially the semantic representations of the world and the core principles of its functioning. LeCun proposes JEPA [12]: "a non-generative architecture for predictive world models that learn a hierarchy of representations". Regardless of how we build this representation or which exact formalism we use, it would be hard to avoid representing objects, their properties, relations and actions: essentially a logic-based representation.

Assuming we have such a representation, the problems of predicting its future states and answering the questions can be tackled in various ways. Our hypothesis is that logic-based reasoning would likely be one of the useful mechanisms, while not the single, and maybe not the most important one. The paper [10] sug-

gests a vision of *LLM-Modulo frameworks*: a tight integration of the strengths of LLMs with external tools, like the *Toolformer* system [23]. The paper [18] reviews works in which language models (LMs) are augmented with reasoning skills and the ability to use tools.

2 Related Work

We will not be able to give a full coverage of the whole field of semantic parsing, containing such classics as CYC [21] etc. Considering modern, prominent semantic parsing frameworks we note Abstract Meaning Representation AMR [3] and Discourse Representation Structures (DRS) [13]. Parsers for both of these frameworks employ machine learning on a large corpora of annotated natural language texts.

While AMR is not built for full first order logic (FOL) with quantifiers, the DRS framework – along with a Combinatory Categorial Grammar (CCG) parser – is behind the powerful Boxer parser [4] and has been connected to FOL reasoners. The UDepLambda parser [22] is similar to ours in the sense that it uses Universal Dependencies [5] as input and derives logical forms of natural language sentences. The LangPro system [1] combines a specialized reasoning engine which builds Lambda Logical Forms (LLFs) from syntactic derivations of Combinatory Categorial Grammar (CCG) and checks a set of linguistic expressions on (in)consistency.

The advent of Large Language Models (LLM)s bootstrapped research on using LLMs for parsing natural language to a symbolic form processable by reasoning tools. The LINC system [19] uses several LLMs as parsers into logic and then employs the Prover9 FOL reasoner [17] for question answering. From the reasoning standpoint, LINC is closest to our system.

The recent papers about AlphaGeometry [30] and Faithful CoT [14] showcase promising examples of converting natural language to a problem-specific symbolic form and then using specialized reasoning, planning and calculation tools to solve the problems presented.

One distinguishing feature of our work is the use of a high-powered full FOL reasoning engine extended with several mechanisms for commonsense reasoning. To our knowledge, no other similar reasoning system exists: in a sense, the closest systems are the Answer Set Programming (ASP) systems, especially the CASP system [2]. The latter is used in the natural language dialog system [8] which relies on GPT-3 for semantic parsing to logic.

3 Representing Concepts for Automated Reasoning

Our starting point is question answering using natural language, with the help of an extended FOL reasoner. Thus we have built a full pipeline containing a semantic parser, a reasoner and a proof verbalizer. The pipeline is intended for conducting experiments with integrating LLMs and specialized machine learning. Since the focus of the current paper is representing the core concepts of natural

language, we will give only a brief overview of the whole pipeline, referring to the papers [28], [27], [26] and [9] for describing the pipeline and the extensions implemented in our FOL reasoner.

3.1 The Question answering Pipeline

The described pipeline is able to handle both the natural language inference (NLI) tasks (given a premise, determine whether a given hypothesis is true, false or indeterminate) and the closely related question answering tasks of finding specific objects which match a given criteria.

For example, the system is able to give an expected answer *"Likely John"* to the sentences along a question *"If an animal likes honey, then it is probably a bear. Most bears are big, although young bears are not big. John is an animal who likes honey. Mike is a young bear. Who is big?"*. As an another example, the system answers *"True"* to the question *"The length of the red car is 4 meters. The length of the black car is 5 meters. The length of the red car is less than 5 meters?"*.

Our system is publicly available at https://github.com/tammet/nlpsolver. The implementation consists of four main software systems. The pipeline driver calls the external Stanza parser [20] from Stanford, giving a Universal Dependencies (UD) [5] graph, then runs the semantic parser on the UD graph, calls the reasoner, and finally builds a natural language answer along with the explanation built from the proofs given by the reasoner.

The parser takes English strings of natural language text as input and outputs extended clausified first-order logic formulas encoded in JSON-LD-LOGIC [29]. The main extension is adding numerical confidences to clauses and implementing default logic by including special literals to encode the exceptions.

Parsing consists of a number of phases, each adding new structural details to the results of the previous phases. For the most part, the phases are implemented procedurally, without using explicit transformation rules and lambda terms in the style of categorical grammars: we found that the more complex aspects of translation cannot be easily expressed with the help of simple transformation rules. In particular, the correct interpretation of a sentence depends heavily on previous sentences and a collected database of objects which have been talked about.

For reasoning the pipeline calls our commonsense reasoner GK, written in C: this is the largest and the most complex part of the pipeline. The GK reasoner uses both the parser output and a selected subset of the world knowledge to solve the questions. Wordnet taxonomies are used to solve the precedence problem of exceptions. Large datasets are parsed, indexed and kept in shared memory for quick re-use. GK is built on top of a conventional high-performance resolution-based reasoner GKC [25] for conventional first order logic. Thus GK inherits most of the capabilities and algorithms of GKC. The main additional features of GK are the following:

– Using a well-known answer clause mechanism for finding a number of different answers, with a configurable limit.

- Finding expected proofs even if a knowledge base is inconsistent. Basically, GK only accepts proofs which contain a clause originating from the question.
- Searching for both a proof of the question and a negation of the question / negation of each concrete answer.
- Estimating the numeric confidence in the statements derived from knowledge bases containing uncertain contrary and supporting evidence obtained from different sources.
- Handling exceptions by implementing default logic via recursively deepening iterations of searches with diminishing time limits.
- Performing reasoning by analogy via employing known similarity scores of words along with exceptions.

There is a separate Python program for regression tests, along with several Python files containing sub-tests, currently over 1600 separate NLI tasks. The pipeline driver is called from a command line, with a natural language text and question as a command line argument, plus a number of optional arguments to control the behaviors like the amount of output.

3.2 Words, Synsets and Compound Nouns

Words are obviously the core part of natural language. Since a word like "bear" can have multiple meanings, a question arises whether to use the literal word directly in the symbolic representation, or to determine the concrete meaning and then use this meaning. In particular, Wordnet [7] uses the notion of a *synset* to refer to a concrete meaning. For example, Wordnet gives two synsets for "bear" as a noun (an animal and an investor with a pessimistic outlook) and a large number of synsets for "bear" as a verb.

A core competence of syntactic parsers like Stanza is to determine the word class (noun, verb, etc). Thus the remaining question of disambiguating the word sense, i.e. determining the suitable synset during parsing, applies to different senses of the same class. We assume that in case we were able to disambiguate the word sense, it would make sense to use the identifier of a synset in the representation. However, papers describing word sense disambiguation systems rarely claim highly accurate results. A simple attempt of using GPT-4 for detecting Wordnet synsets reveals that although GPT-4 understands the general pattern of synset identifiers, it does not excel in determining the correct synset. Moreover, all the large commonsense knowledge bases we know of also use literal words, not synsets. Hence our pipeline uses literal words. Fortunately, English words with multiple meanings tend to have one dominating meaning and thus word sense disambiguation is not a critical problem in the context of solving relatively simple question answering tasks.

When we say "uses literal words" above, we really mean lemmas of the words as determined by the Stanza parser: "bear" instead of "bears", "eat" instead of "ate" etc. The same principle appears to be commonly used in large commonsense knowledge bases. For example, Quasimodo contains triplets like "elephant, be in, asia". Using lemmas instead of unmodified words allows us to write more

abstract rules: for example, instead of having separate rules for the specific consequences of "eat" and "ate", it suffices to have rules for "eat". The additional morphological information carried by the original word – typically the tense and singular/plural distinction – is lost in the triplets like the one above, but encoded in parts of the representation used by our pipeline, to be discussed later.

Consecutive nouns like "trash can" or "baby bear" are sometimes identified by Wordnet as a single concept or open compound noun ("trash can", "building material") and sometimes not ("baby bear", "research project"). The Stanza parser has a tendency to identify any consecutive nouns as compound nouns (all of the examples above, as well as "sun bottle", "moon trigonometry" etc). The main question for question answering is whether both or one of the compound nouns in the sentence hold alone. For example, we would want to answer "yes" to the questions "John is a baby bear. John is a bear? John is a baby?" while "unknown" to the questions "John has a trash can. John has trash? John has a can?". Currently our pipeline blatantly disregards the possibility that an open compound noun could be an atomic concept and thus answers "yes" to both kinds of questions above. However, it would not be hard to rectify this issue for common cases by simply checking whether a compound noun is present in Wordnet as a separate synset.

3.3 Proper nouns, Objects, Coreference and a Global ID

Proper nouns like "John" and "New York" function as labels for concrete objects, either real or imaginary. The simplest way to represent these is to generate a new constant symbol for each new proper noun encountered. The alternative potential approach of generating $\exists$ `X has_name(X, John)` ... is worse: it would lead to a new constant symbol after Skolemization anyway and would push the coreference problem recalled below to be solved during proof search instead of parsing.

Obviously, we need to detect whether a noun in the parsed text functions as a label for something we have labelled before. Say, in "John Smith received a prize. John was really happy. He had wanted it so much." we expect both "John" and "He" to be represented by the same constant as "John Smith". Similarly, "animal" in the sentences "The bear ate a fish. The animal was hungry" should be represented by the same constant as "The bear".

This *coreference problem* is one of the classic questions of semantic parsing, famously hard to solve. Our system uses a set of heuristics to give acceptable results for relatively simple cases like the ones above. During parsing we build up a database of encountered objects along with the properties detected without further complex inferences, and then match the new labels to the database, giving bonuses for recent objects. However, our experiments show – unsurprisingly – that with the help of suitable multishot prompting, GPT-4 exhibits a much better performance than our set of simple heuristics. Although the version 1.7 of the Stanza parser contains a specialized transformer-based coreference solution model, we have not yet had an opportunity to compare its performance with the multi-shot prompt engineering approach for GPT-4.

The concrete approach taken in our pipeline regarding the naming of constants for proper nouns is to generate a new constant like `c2_John` or `the_c3_car`. This approach creates problems when composing the symbolic representation of one text with a representation of another: there is no obvious way to detect that a local identifier/constant in one should be equal to a constant in another. Probably the best way to solve the issue is to use *wikification*, i.e. representing objects by global identifiers: the URLs of the corresponding Wikipedia article, if available. In our experiments with multishot LLM-based parsing we additionally ask GPT-4 to perform wikification, with good results.

3.4 Complex concepts, Phrases and Triplets

Moving further from the issue of open compound nouns discussed earlier, we come to the question of representing complex concepts like those found in typical large triple stores or knowledge graphs of commonsense knowledge. Let us consider some examples of such concepts in Quasimodo triplets ("bear, has_body_part, tail", "bear, get stung by, bees", "bear, choke on, fish bones", "bear, know, when to wake up"), Nell triplets ("concept:musicianplaysinstrument, inverse, concept:instrumentplayedbymusician"), UnCommonSense ("penguin, NotCapableOf, eat food", "penguin, NotReceivesAction, threatened with extinction"), ConceptNet ("bear, is a type of, a sacred animal", "bear, is capable of, winter over in his den"), Wikidata ("Winterthur, instance of, municipality of Switzerland", "Winterthur, twinned administrative body, Ontario").

In all these cases the reason for the use of complex multi-word concepts as atomic forms is the reliance on the triplet or knowledge graph format to represent complex information. The object-property-value triplet provides very little structure to represent information (no quantifiers, connectives, high-arity relations, etc), thus any nontrivial piece of knowledge has to be either represented via large chunks of verbatim or modified phrases like in the examples above, or – potentially – via reification and the recursive introduction of new identifiers for smaller and smaller parts of phrases. The latter option would likely lead to encoding in some form of logic with the help of triplets: it is certainly possible, just as it is possible to encode everything by using just binary trees, but unlikely to be beneficial when compared to more conventional encoding structures. None of the large triplet-based commonsense knowledge bases we know of uses this option.

Using triplets and atomic complex phrases in triplets is well-suited for human presentation formats like Wikipedia info-boxes, but ill-suited for asking questions and using rules. For example, knowing "Winterthur, instance of, municipality of Switzerland" does not allow us to directly and mechanically derive that Winterthur is a municipality or that it is located in Switzerland. Suppose we want to mechanically infer from a hypothetical triplet containing variables X and Y as in "X, instance of, municipality of Y" that "X, in, Y" should hold. We would need to first split the phrase "municipality of Y" in our representation into components like `true_phrase_3("municipality","of",Y)`. Second, we would need to represent common phrases like "instance of", "is a", "was a" etc with a single

identifier. Third, we would need to attach quantifiers to variables to indicate that they are indeed variables and whether they stand for something general or indicate the existence of something. Finally, we have to indicate that from one statement we can infer another. All in all, we will arrive to an essentially logic-based rule like $\forall\, \mathtt{X}, \mathtt{Y}(\mathtt{isa}(\mathtt{X}, \mathtt{phrase}(\text{"}\mathtt{municipality}\text{"}, \text{"}\mathtt{of}\text{"}, \mathtt{Y})) \rightarrow \mathtt{in}(\mathtt{X}, \mathtt{Y}))$

3.5 Relation wrappers

In textbooks, the most common representation of class membership, relation or property like in "John is a bear. John is a father of Mike. Mike is green." looks like $\mathtt{bear}(\mathtt{John})$ & $\mathtt{father}(\mathtt{John}, \mathtt{Mike})$ & $\mathtt{green}(\mathtt{Mike})$, with English words translated to predicate symbols. This is essentially the same representation we get from GPT-4 with a simple prompt "You are a semantic parser from English to first order predicate logic (FOL). Convert input sentences to logic.".

In first order logic, this representation does not allow us to quantify over words. For example, in RDFS [16] the $\mathtt{rdfs:domain}$ relation indicates that the rightmost argument of the relation belongs to a certain class. As an example, the $\mathtt{movieactor}$ relation could be declared to have a domain of a class $\mathtt{movie}$. A naive rule

$$\forall\, \mathtt{X}, \mathtt{R}, \mathtt{Z}, \mathtt{U}\ (\mathtt{R}(\mathtt{X}, \mathtt{Z})\ \ \&\ \ \mathtt{rdfs:domain}(\mathtt{R}, \mathtt{U}) \rightarrow \mathtt{rdf:type}(\mathtt{Z}, \mathtt{U}))$$

quantifies over a predicate symbol $\mathtt{R}$, which is not allowed in FOL. Therefore, translating RDF triplets to FOL requires the use of a *wrapping predicate* like $\mathtt{rdf}$ to wrap the whole triplet. Rewriting the incorrect rule above by introducing the wrapping predicate gives us

$$\forall\, \mathtt{X}, \mathtt{R}, \mathtt{Z}, \mathtt{U}\ (\mathtt{rdf}(\mathtt{R}, \mathtt{X}, \mathtt{Z})\ \ \&\ \ \mathtt{rdf}(\mathtt{rdfs:domain}, \mathtt{R}, \mathtt{U}) \rightarrow \mathtt{rdf}(\mathtt{rdf:type}, \mathtt{Z}, \mathtt{U})).$$

Note that here and in the following we will use a convention that the first argument of the wrapper is a translation of the English word indicating the relation, property or action.

Similarly, in case we want to write a general rule of transitivity allowing us to derive "John is nicer than Mike" from "John is nicer than Eve. Eve is nicer than Mike" we would need a wrapper like in

$$\forall\, \mathtt{R}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\ (\mathtt{relation}(\mathtt{R}, \mathtt{X}, \mathtt{Y})\ \ \&\ \ \mathtt{relation}(\mathtt{R}, \mathtt{Y}, \mathtt{Z}) \rightarrow \mathtt{relation}(\mathtt{R}, \mathtt{X}, \mathtt{Z}).$$

Since not all relations are transitive, such a rule would be conceptually wrong in the general case. One option to overcome the problem is to write rules like this with a qualifier like in

$$\begin{aligned}&\forall\, \mathtt{R}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\ (\mathtt{transitive\_word}(\mathtt{R}) \rightarrow \\ &\quad (\mathtt{relation}(\mathtt{R}, \mathtt{X}, \mathtt{Y})\ \ \&\ \ \mathtt{relation}(\mathtt{R}, \mathtt{Y}, \mathtt{Z}) \rightarrow \mathtt{relation}(\mathtt{R}, \mathtt{X}, \mathtt{Z}))\end{aligned}$$

Another – the one we use – is to use a larger number of wrappers, indicating the specific kind of a relation. More concretely, we use prepositions to determine

the relation type and use `rel2_than` to wrap the *nice* relation in "John is nicer than Eve", `rel2_of` to wrap the *father* relation in "John is a father of Mike" and just `rel2` to wrap the *in* relation in "John is in a house". This makes it possible to keep the information conveyed by prepositions like *of*, thus enabling us to write different rules for different relation classes. Essentially, we solve the question of whether a relation is transitive already during the semantic parsing phase, instead of enforcing the reasoning engine to determine the validity of atoms like `transitive_word(R)` during search.

3.6 Properties, Class membership, Taxonomies

In our understanding the main difference between object properties – typically indicated by adjectives like "big", "green", "nice" etc – and class membership ("Jonathan is a seagull": Jonathan belongs to the "seagull" class, "This car is nice": there is an object belonging to the "car" class) is the relative permanence and context-insensitivity of class membership, plus a tendency to belong to a deep taxonomy of classes. Indeed, seagulls never turn into people or cars, and vice versa. However, permanence is not absolute for all classes: "Mike is a banker" indicates a less permanent class membership than "seagull". We could possibly even argue that "banker" should be considered a property, not a class. For time being, our system assumes permanence of class membership and thus we encode the "seagull" sentence above simply as `isa(seagull, c1_Jonathan)`.

In contrast, properties like "green" are relatively easy to change (for example, by painting a green object red), may have varying degrees of intensity ("very green", "slightly green", "extremely big", "somewhat big") and may be relative: "a big mouse" is not necessarily a "big" object. Consider the sentences "Albert is a big mouse. Mike is an elephant. Elephants are big. Mice are small.". Arguably we would answer "yes" to the question "Mike is big?", "no" to the question "Albert is big?" (although this is somewhat ambivalent) and "yes" to the question "Albert is a big mouse?".

Thus the question: how to represent such qualities of a property? One option would be to use the analogue of Davidsonian semantics of verbs [15], but for properties: assigning a new identifier to a property and then indicating the specific qualities for this identifier. For example, "Albert was a very small mouse" could be encoded using `p1` as an identifier of the property and stating

$$\texttt{isa}(\texttt{mouse}, \texttt{c1\_Albert}) \ \& \ \texttt{has\_property}(\texttt{p1}, \texttt{c1\_Albert}) \ \&$$
$$\texttt{is\_property}(\texttt{small}, \texttt{p1}) \ \& \ \texttt{relative\_to\_class}(\texttt{mouse}, \texttt{p1}) \ \&$$
$$\texttt{has\_intensity}(\texttt{high}, \texttt{p1}) \ \& \ \texttt{has\_time}(\texttt{past}, \texttt{p1})$$

However, since our goal is to support efficient question answering using logic, we take the opposite approach: combine all the potential qualities into one atom, with argument positions indicating the type of the quality. The main gain is shortening the derivations: a proof answering the corresponding question "Albert was a very small mouse?" becomes much shorter than would be the proof using the Davidsonian-like semantics. While finding a long proof could be OK for this

specific question, any derivations of qualified properties during more complex proofs would become longer, with the most unwelcome side effect of exploding the search space, i.e. making it much harder to find a proof. In general, the complexity of finding a proof in FOL is undecidable, i.e. it has no ordinary complexity measure: we cannot say exactly how much harder does the proof search become.

Thus we encode the sentence above as

$$\texttt{isa}(\texttt{mouse},\texttt{c1\_Albert}) \ \& \ \texttt{prop}(\texttt{small},\texttt{c1\_Albert},3,\texttt{mouse},\$\texttt{ctxt}(\texttt{Past},1))$$

while encoding a simpler sentence "Albert was small" as

$$\texttt{prop}(\texttt{small},\texttt{c1\_Albert},\$\texttt{generic},\$\texttt{generic},\texttt{ctxt}(\texttt{Past},1))$$

where 1 encodes "low", 3 encodes "high" and $\texttt{\$generic}$ essentially means "not given" and the arguments of the `prop` predicate are: property name, object having a property, intensity of the property, relative class of the property, context. The encoding of intensity and context requires further discussion. First, our intuition (without any experimental grounds) is that the expressed intensity of a property in normal circumstances is one of three: either unknown (often, but not always indicating "average"), high or low. Excluding subtle cultural differences and stylistic issues, the qualifiers like "very", "extremely", "super", "hugely" all appear to mean the same thing: clearly over the average. We cannot really say whether "very" indicates a lower or higher intensity than "extremely" or "hugely". We have the same intuition for low intensity qualifiers like "a bit", "somewhat", "sort of" etc. In order to enable answering "yes" to the question "Albert was a very small mouse. Albert was a small mouse?" while answering "no" to "Albert was a somewhat small mouse. Albert was a very small mouse?" and "unknown" to the questions in "Albert was a small mouse. Albert was a very small mouse? Albert was a somewhat small mouse?" we axiomatize the relations between properties of different intensities with the following formulas:

$$\forall\, \texttt{W},\texttt{X},\texttt{C},\texttt{Y}\ (\texttt{prop}(\texttt{W},\texttt{X},3,\texttt{C},\texttt{Y}) \rightarrow \texttt{prop}(\texttt{W},\texttt{X},\$\texttt{generic},\texttt{C},\texttt{Y})$$
$$\forall\, \texttt{W},\texttt{X},\texttt{C},\texttt{Y}\ (\texttt{prop}(\texttt{W},\texttt{X},1,\texttt{C},\texttt{Y}) \rightarrow \texttt{prop}(\texttt{W},\texttt{X},\$\texttt{generic},\texttt{C},\texttt{Y})$$
$$\forall\, \texttt{W},\texttt{X},\texttt{C},\texttt{Y}\ (\texttt{prop}(\texttt{W},\texttt{X},1,\texttt{C},\texttt{Y}) \rightarrow \neg\texttt{prop}(\texttt{W},\texttt{X},3,\texttt{C},\texttt{Y})$$

Regarding taxonomies – a fundamental part of most ontologies – we have essentially two options: either use atomic formulas like

$$\texttt{subclass}(\texttt{jeep},\texttt{car}),\ \texttt{subclass}(\texttt{car},\texttt{motor\_vehicle}),\ldots$$

etc along with (at least) the rules

$$\forall\, \texttt{X},\texttt{Y},\texttt{Z}\ (\texttt{subclass}(\texttt{X},\texttt{Y}) \ \& \ \texttt{isa}(\texttt{Z},\texttt{X}) \rightarrow \texttt{isa}(\texttt{Z},\texttt{Y}))$$
$$\forall\, \texttt{X},\texttt{Y},\texttt{Z}\ (\texttt{subclass}(\texttt{X},\texttt{Y}) \ \& \ \texttt{subclass}(\texttt{Y},\texttt{Z}) \rightarrow \texttt{subclass}(\texttt{Y},\texttt{Z}))$$

or give direct rules for all the subclass relations, like

$$\forall\, \texttt{X}(\texttt{isa}(\texttt{jeep}, \texttt{X}) \rightarrow \texttt{isa}(\texttt{car}, \texttt{X}))$$

In our implementation we have currently opted for the second version, where the subclass transitivity rule is not needed. However – importantly – our reasoner has a built-in subsystem for quickly checking whether one class is a subclass of another, following the Wordnet taxonomy hierarchy. This mechanism is primarily important for efficiently handling default logic with priorities, i.e. layered exceptions to rules.

3.7 Context

Properties, relations and actions are generally not permanent or absolute. Typically they depend on time, while sometimes also on location, viewpoint or source. We call all such parameters a *context* and collect them as arguments in a *context term* like $\texttt{\$ctxt}(\texttt{Past}, 1)$ in the property representation

$$\texttt{prop}(\texttt{small}, \texttt{c1\_Albert}, 3, \texttt{mouse}, \texttt{\$ctxt}(\texttt{Past}, 1))$$

above. The question of which parameters should be considered – and what are the best ways to represent them – is nontrivial. Hence our system is currently representing only time, and even this with broad strokes only: we indicate *Past* and *Present* along with a growing counter of situation-changing actions. In order to use the system for creating action plans – and optimizing planning search – we should also add the action operators into the context. We have not yet experimented with using GPT-4 for constructing a complex context from the input text.

A potential alternative to the choice of collecting all the context parameters into a single term would be adding them directly as arguments of wrappers like `prop` or `rel2`. This would, however, force us to quantify over all these parameters separately, whenever a statement does not depend on the context. It would make it more cumbersome to axiomatize the equality of different contexts. It would also make the management of a knowledge base harder. Essentially, collecting the parameters into a context term is a pragmatic choice to make the context a first class object on its own.

3.8 Selecting quantifiers and Skolemization

The representation of sentences describing concrete situations does not need quantifiers: we will simply generate constants for objects and produce the logical forms of their properties, relations and actions. Arguably, most semantic parsers intentionally limit their functionality to parsing such sentences.

Parsing sentences stating general *rules* like "Bears are large animals" to a symbolic form like

$$\begin{aligned}\forall\, \texttt{X}\ (&\texttt{isa}(\texttt{bear}, \texttt{X}) \rightarrow (\texttt{isa}(\texttt{animal}, \texttt{X})\ \ \&\\ &\texttt{prop}(\texttt{large}, \texttt{X}, \texttt{\$generic}, \texttt{bear}, \texttt{\$ctxt}(\texttt{Present}, 1))))\end{aligned}$$

requires quantifiers. In this example it looks fairly obvious that we should use a general quantifier. At the same time, parsing "Bears liked to swim" should generate an existential quantifier or a constant for representing a concrete set of bears. Moreover, in sentences like "Bears have a tail" we expect the variable for bears to be generally quantified, while – since each bear has its own tail – the variable for a tail has to be existentially quantified in the scope of the general quantifier

$$\forall\, \texttt{X}\ (\texttt{isa}(\texttt{bear}, \texttt{X}) \rightarrow \exists\, \texttt{Y}\ (\texttt{isa}(\texttt{tail}, \texttt{Y})\ \ \&\ \ \texttt{rel2}(\texttt{have}, \texttt{X}, \texttt{Y}, \$\texttt{ctxt}(\texttt{Present}, 1))))$$

or, better yet, generalizing over all contexts:

$$\forall\, \texttt{X}\ (\texttt{isa}(\texttt{bear}, \texttt{X}) \rightarrow \exists\, \texttt{Y}\, \forall\, \texttt{Z}\ (\texttt{isa}(\texttt{tail}, \texttt{Y})\ \ \&\ \ \texttt{rel2}(\texttt{have}, \texttt{X}, \texttt{Y}, \texttt{Z})))$$

Skolemizing the tail variable `Y` gives us a new function term `cs1(X)` with a new function symbol `cs1` representing the conceptual operation of selecting the tail of a specific bear.

However, a sentence "Americans have a capital" should generate an existential quantifier for the "capital" variable, out of the scope of the universal quantifier for the variable for americans. The first sentence of the Wikipedia article for a fork starts with "In cutlery or kitchenware, a fork (from Latin: furca 'pitchfork') is a utensil, ..." which could be erroneously interpreted as speaking about a concrete fork.

As these examples show, quantifier selection is a complex question. Like in the case of coreference resolution, our system uses a set of heuristics to determine whether and which quantifiers to use: an approach which works well for relatively easy cases, but often fails for edge cases. Again, just like with coreference resolution, our experiments show that a suitable prompt with multiple examples (multi-shot) makes GPT-4 to give correct results more often than our heuristics.

3.9 Verbs and actions

Like nouns, verbs may have different meanings, with PropBank [11] and VerbNet [24] being the widely used resources for classifying and identifying their different meanings. Although we plan to use these resources, not the least for gleaning useful axioms from the ones provided by VerbNet, our system currently uses literal verbs instead of the identifiers of concrete meanings.

Verbs are related to action frames and tend to have a larger number and more varied parameters than adjectives: actor (agent), target (patient), destination, duration, co-agent etc. Thus Davidsonian semantics of events [15] suggests assigning a new identifier to each event and then indicating the specific parameters and qualities for this identifier. While we do not follow this approach for adjectives (properties of nouns), we use a semi-Davidsonian approach for verbs. We collect the verb itself, actor, target, event identifier and the context into a single atom, with the same goal of making proof search more efficient by shortening the proofs. The other, less common parameters are connected with the event

identifier in separate atoms. For example, the main part of the representation of "John quickly ate the apple" is

$$\texttt{act2}(\texttt{eat}, \texttt{c1\_John}, \texttt{the\_c2\_apple}, \texttt{cs3}, \$\texttt{ctxt}(\texttt{Past}, 1)) \ \& \\ \texttt{prop}(\texttt{quickly}, \texttt{cs3}, \$\texttt{generic}, \$\texttt{generic}, \$\texttt{ctxt}(\texttt{X}, 1))$$

where `cs3` is the event identifier. For actions without the target (patient) like in "John ate" we use the wrapper predicate `act1` without the target argument. While verbs typically denote concrete actions, they can also be used in a rule-like manner as in:

- "Birds fly", denoting a typical activity. We use the wrappers `do1` and `do2` to differentiate from concrete actions.
- "Birds can fly", denoting a capability to perform an activity. We use the wrappers `can1` and `can2` to differentiate from the ones above.

As an example of axioms relating the meaning of these wrappers we bring

$$\forall\, \texttt{W}, \texttt{X}, \texttt{Z}, \texttt{C}\ (\texttt{act1}(\texttt{W}, \texttt{X}, \texttt{Z}, \texttt{C}) \rightarrow \texttt{can1}(\texttt{W}, \texttt{X}, \texttt{Z}, \texttt{C})) \\ \forall\, \texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}, \texttt{C}\ (\texttt{act2}(\texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}, \texttt{C}) \rightarrow \texttt{can1}(\texttt{W}, \texttt{X}, \texttt{Z}, \texttt{C})) \\ \forall\, \texttt{W}, \texttt{X}, \texttt{Z}, \texttt{C}\ (\texttt{do1}(\texttt{W}, \texttt{X}, \texttt{Z}, \texttt{C}) \rightarrow \texttt{can1}(\texttt{W}, \texttt{X}, \texttt{Z}, \texttt{C}))$$

enabling the system to answer "John" to the question "John drove a car. Who can drive?" and "yes" to the question "Birds fly. Mike is a bird. Mike can fly?"

3.10 Confidences, Exceptions, Measures and Sets

The main extensions to FOL implemented by our reasoner are numeric measures of subjective confidence and exceptions to rules, the latter based on default logic.

Confidences allow the system to estimate the numeric confidence in the statements derived from knowledge bases containing uncertain contrary and supporting evidence obtained from different sources. Exceptions are handled by implementing default logic with precedences via recursively deepening iterations of searches with diminishing time limits. Given sentences "Penguins are birds. Penguins cannot fly. Birds can fly. John is a bird. Mike is a penguin." the system answers "John" to the question "Who can fly?" and "Mike" to the question "Who cannot fly?".

Our parser estimates confidence numbers based on the presence of either direct probabilities ("John smokes with a probability 90%") or quantifiers like "most", "some" etc ("Most birds can fly"). Every clause in the final clause form of the logical representation is attached a confidence number and the inference operations calculate the confidences of resulting clauses, based on the principles presented in our paper [27]. Sentences expressing rules like "Birds can fly", unless strengthened ("All birds can fly", "Every bird can fly") are enriched with the special blocking literal, the derivability of which will cause the clause containing the literal to be discarded. The details of implementing default logic can be found in our paper [26].

We employ the built-in arithmetic and equational reasoning of GK for solving simple numerical problems, like comparing costs or other measures. For example, the relevant parts of the representation of sentence "The book costs 10 dollars" are: $\texttt{rel2}(\texttt{have}, \texttt{the\_c1\_book}, \mathit{term}, \$\texttt{ctxt}(\texttt{Pres}, 1)) \ \& \ = (10, \$\texttt{count}(\mathit{term}))$ where *term* is $\texttt{measure1}(\texttt{price}, \texttt{the\_c1\_book}, \texttt{dollar}, \$\texttt{ctxt}(\texttt{Pres}, 1))$. We have also started initial work on representing sets and using set operations.

4 Experiments with using LLMs as a Part of the Pipeline

While our hand-coded semantic parser is capable of adequately representing relatively simple sentences, it is clear that further improvements without incorporating machine learning would not be cost-effective. A possible approach used by several parsers is specialized training on annotated corpora. However, it is highly probable that using pre-trained LLMs as a basis would give higher quality and incur lower development costs. So far we have conducted experiments with GPT-4 in two directions, both with the most promising results.

The first direction is building a pipeline of specialized tasks with prompt engineering: solving coreference problems and wikification, determining the suitable quantifiers, and most importantly, simplifying complex sentences to a point where they become adequately parsable by our system. While we use N-shot prompting with ca 20-30 examples for the first two, we use the following zero-shot prompt for simplification: ' 'You preprocess sentences for a semantic parser to logic. Simplify, maximally shorten and split the sentence to shortest possible separate subsentences, to make it understandable for children.".

The second direction is focusing on direct translation of natural language to FOL, encoded in JSON and using the conceptual principles outlined in the current paper. We have observed that relatively simple prompts without a significant number of examples typically lead to low quality, essentially useless output. However, our experiments with ca 100 and more carefully selected examples in the prompt have already led to a much higher quality output than we could expect from further hand-coded improvements of our current parser. This said, the GPT-4 output is certainly not perfect. Our current hypothesis is that combining several approaches with specialized post-processing of the GPT output has the most promise. Our experimental prompts are available in https://github.com/tammet/nlpsolver/tree/main/gpt.

5 Summary and Future Work

We have described the principles of representing natural language concepts in FOL, geared towards efficient handling by FOL provers. Our pipeline is able to use this representation to solve nontrivial commonsense questions by using our commonsense-extended prover GK. We have built a regression test set of over 1600 natural language reasoning problems. The main lines of our current work are following:

- Incorporating LLMs into the semantic parsing pipeline.
- Improving the capabilities of the conceptual representation scheme and the reasoning engine for sets, counting, arithmetic and databases.
- Incorporating specialized statistical and machine learning components to the GK reasoner, in order to improve the efficiency in case a large commonsense knowledge base is given as a background.
- Using the system as a part of the process of automatic generation of background knowledge from existing knowledge graphs and natural language texts like Wikipedia articles.

References

1. Abzianidze, L.: Langpro: Natural language theorem prover. arXiv preprint arXiv:1708.09417 (2017)
2. Arias, J., Carro, M., Salazar, E., Marple, K., Gupta, G.: Constraint answer set programming without grounding. Theory and Practice of Logic Programming **18**(3-4), 337–354 (2018)
3. Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., Schneider, N.: Abstract meaning representation for sembanking. In: Proceedings of the 7th linguistic annotation workshop and interoperability with discourse. pp. 178–186 (2013)
4. Curran, J.R., Clark, S., Bos, J.: Linguistically motivated large-scale nlp with c&c and boxer. In: Proceedings of the 45th annual meeting of the Association for Computational Linguistics Companion volume proceedings of the demo and poster sessions. pp. 33–36 (2007)
5. De Marneffe, M.C., Manning, C.D., Nivre, J., Zeman, D.: Universal dependencies. Computational linguistics **47**(2), 255–308 (2021)
6. Dziri, N., Lu, X., Sclar, M., Li, X.L., Jiang, L., Lin, B.Y., Welleck, S., West, P., Bhagavatula, C., Le Bras, R., et al.: Faith and fate: Limits of transformers on compositionality. Advances in Neural Information Processing Systems **36** (2024)
7. Fellbaum, C.: WordNet: An electronic lexical database. MIT press (1998)
8. Gupta, G., Zeng, Y., Rajasekaran, A., Padalkar, P., Kimbrell, K., Basu, K., Shakerin, F., Salazar, E., Arias, J.: Building intelligent systems by combining machine learning and automated commonsense reasoning. In: Proceedings of the AAAI Symposium Series. vol. 2, pp. 272–276 (2023)
9. Järv, P., Tammet, T., Verrev, M., Draheim., D.: Knowledge integration for commonsense reasoning with default logic. In: Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - KEOD. pp. 148–155. INSTICC, SciTePress (2022)
10. Kambhampati, S., Valmeekam, K., Guan, L., Stechly, K., Verma, M., Bhambri, S., Saldyt, L., Murthy, A.: Llms can't plan, but can help planning in llm-modulo frameworks. arXiv preprint arXiv:2402.01817 (2024)
11. Kingsbury, P.R., Palmer, M.: From treebank to propbank. In: LREC. pp. 1989–1993 (2002)
12. LeCun, Y.: A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. Open Review **62**(1) (2022)
13. Liu, J., Cohen, S.B., Lapata, M.: Discourse representation structure parsing. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 429–439 (2018)

14. Lyu, Q., Havaldar, S., Stein, A., Zhang, L., Rao, D., Wong, E., Apidianaki, M., Callison-Burch, C.: Faithful chain-of-thought reasoning. arXiv preprint arXiv:2301.13379 (2023)
15. Maienborn, C.: Event semantics. Edited by Claudia Maienborn Klaus von Heusinger p. 232 (2011)
16. Manola, F., Miller, E., McBride, B., et al.: Rdf primer. W3C recommendation **10**(1-107), 6 (2004)
17. McCune, W.: Release of prover9. In: Mile high conference on quasigroups, loops and nonassociative systems, Denver, Colorado (2005)
18. Mialon, G., Dessì, R., et al.: Augmented language models: a survey. CoRR abs/2302.07842 (2023)
19. Olausson, T.X., Gu, A., Lipkin, B., Zhang, C.E., Solar-Lezama, A., Tenenbaum, J.B., Levy, R.: Linc: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. arXiv preprint arXiv:2310.15164 (2023)
20. Qi, P., Zhang, Y., Zhang, Y., Bolton, J., Manning, C.D.: Stanza: A python natural language processing toolkit for many human languages. CoRR abs/2003.07082 (2020), https://arxiv.org/abs/2003.07082
21. Ramachandran, D., Reagan, P., Goolsbey, K.: First-orderized researchcyc: Expressivity and efficiency in a common-sense ontology. In: AAAI workshop on contexts and ontologies: theory, practice and applications. pp. 33–40 (2005)
22. Reddy, S., Täckström, O., Petrov, S., Steedman, M., Lapata, M.: Universal semantic parsing. arXiv preprint arXiv:1702.03196 (2017)
23. Schick, T., Dwivedi-Yu, J., Dessì, R., et al.: Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems **36** (2024)
24. Schuler, K.K.: Verbnet: extensions and mappings to other lexical resources. In: 2007-01-04]. http://www. coll. uni-saarland. de/projects/salsa/workshop/contents/workshop _ slides/slides4. pdf (2006)
25. Tammet, T.: GKC: A reasoning system for large knowledge bases. In: Fontaine, P. (ed.) Proc. of CADE'2019 – the 27th Intl. Conf. on Automated Deduction. LNCS, vol. 11716, pp. 538–549. Springer (2019)
26. Tammet, T., Draheim, D., Järv, P.: Gk: Implementing full first order default logic for commonsense reasoning (system description). In: Blanchette, J., Kovács, L., Pattinson, D. (eds.) IJCAR 2022: Automated Reasoning. LNCS, vol. 13385, pp. 300–309. Springer (2022)
27. Tammet, T., Järv, P., Draheim, D.: Confidences for commonsense reasoning. In: Platzer A., S.G. (ed.) Automated Deduction – CADE 28. CADE 2021. LNCS, vol. 12699, pp. 507–524. Springer (2021)
28. Tammet, T., Järv, P., Verrev, M., Draheim, D.: An experimental pipeline for automated reasoning in natural language (short paper). In: International Conference on Automated Deduction. pp. 509–521. Springer (2023)
29. Tammet, T., Sutcliffe, G.: Combining json-ld with first order logic. In: 15th International Conference on Semantic Computing (ICSC). pp. 256–261. IEEE (2021)
30. Trinh, T.H., Wu, Y., Le, Q.V., He, H., Luong, T.: Solving olympiad geometry without human demonstrations. Nature **625**(7995), 476–482 (2024)
31. Valmeekam, K., Marquez, M., Olmo, A., Sreedharan, S., Kambhampati, S.: Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. Advances in Neural Information Processing Systems **36** (2024)